# Team 1 MP Report

Jessica Zhu, Yu Bu, Trent Zhang

October 2021

## 1 System Design

Our dashboard was written in Python and the application is hosted by the Python Dash framework. When the application is running, it can be accessed at *localhost:8050* in the browser; more specific setup instructions are included in our GitHub repository's ReadMe file.

At a high-level, the linked Reddit dataset for Scenario 3 was fed into three data models of our choice: a relational database (MySQL), a document-oriented store (MongoDB), and a graph database (Neo4j). Because the application is purely hosted locally, it may not appear to work correctly if the databases are not set up. The design consists of a few widgets, which will be further explained in Section 2. Briefly, one widget allows the user to configure and create workflows. Another widget displays various attributes of user-created workflows and allows the user to initiate a workflow of their choice. Finally, the last widget allows the user to visualize the data in their selected database and table.

When a workflow is started, the first step of the workflow process is fetching results that match the user's workflow. Closely-matched results are presented in a table to the user and they can choose which results to keep or remove; this is our feature for human-in-the-loop interaction. After the user chooses what results to keep or remove, the workflow continues.

## 2 Functionalities

Current working functionalities are demonstrated in our demo video, but this section will also go over them.

First, we present a small widget that allows a user to select the supported database of their choice. This database is the one on which the workflow is run, and the one on which a later widget fetches data from for the user to view at their leisure. If MySQL is the selected database, a query field is also available to the user; they can input the query of their choice, which is fed into MySQL, and the results are displayed back to the user. This is currently not available for MongoDB or Neo4j, as we believed it would be difficult for a user to remember their syntax, but could be added in. Regardless of the database, though, the data is displayed to the user. For the databases, only a certain amount of results are shown; we found that trying to load the entire Reddit dataset into the application made load time very slow and unreasonable for the user. The data is populated into a table, and the user can select a cell of their choice and the full text will be displayed to the user (in case the cell is too small for the full message, especially for the Reddit comments).

Next, we have a widget that consists of a form, allowing the user to create a workflow. When the Create Workflow button is clicked, the workflow is added to a table of workflows presented in a separate widget. We are aware that the placeholder text of this form may not fully appear and hope to change that in the future. The form has many optional input fields; the user can specify the name of the workflow, the minimum comment score, the maximum controversiality score, the author, any search words within the comments, how often the workflow should be executed in minutes, and any dependencies - i.e. should the created workflow run after another created workflow has run. For example, the user could specify that they only want to see comments made by Reddit user ”exampleUser”, the comment score must be greater than 5, the comment text must contain ”sick” somewhere in it, and this workflow executes after workflow 2 has completed.

Another widget displays a table of workflows, initially empty until a workflow is created from the above form. Once the table is populated, the user can select any table cell and initiate the workflow corresponding to it. The workflow will then begin executing. The Status column of the corresponding workflow will change as the workflow executes. The first step of the workflow process is querying the database for data that matches the given conditions inputted by the form. The results strictly matching the conditions are saved, but the results closely matching the conditions are displayed to the user as a table. The user can select data from the closely matching results to save to the table (i.e. if they think the result is ”good enough”). Afterwards, any data that strictly matched the conditions *and* any selected data from the loosely matching results are saved as a new table into the corresponding data model; for example, if the workflow is run in MySQL, a new table representing the workflow results is created in the MySQL database. On the other hand, if the workflow is run in MongoDB, a new collection representing the workflow is created. Once these tables are created, they can be viewed in the data visualization widget (which by default shows the Reddit dataset, but there is a dropdown allowing the user to select another table in the database of their choice).

The system was designed with Scenario 3 in mind. A doctor could create a workflow to query the database for a keyword such as ”illness” or ”remedy” to find popular home remedies used by redditors. Although we used the provided Reddit dataset, we wanted to note that this dataset did not provide many comments related to healthcare in any way, so it was not very sufficient to the specific scenario. However, a larger dataset (and perhaps more specialized, such as on subreddits dedicated to alternative treatments) would be much more helpful. Due to time constraints, we did not fill the database with synthetic data beyond what was included in the original Reddit dataset.

Our advanced features are as follows:

1. Implementation of human-in-the-loop intervention into the workflow
2. Allowing workflows to execute interdependently rather than only in a scheduled manner
3. Timely updated visualized tables and queuing workflow traffic

## 3 Challenges

One of the many challenges with developing this dashboard was the problem of offering multi-data model support. Because of how differently most of the data models are structured, this required writing different code cases for each data model. For example, while it is reasonable for MongoDB and MySQL to display the results of their collections and tables respectively, Neo4j is a graphical database and it is a more complex problem: to create a Pandas dataframe out of the graphical query results.

Creating a workflow result table in the database was also harder than expected, because the syntax to create a new table is drastically different between MongoDB, MySQL, and Neo4j. Different code had to be written for each case. Not only that, Neo4j has no notion of separate tables and collections, and we had to get a bit creative with displaying the Neo4j workflow results. In Neo4j's case, we didn't save the resultant table into Neo4j database since Neo4j does not support tables. Instead, we create a file of the results into a Neo4j folder on the local machine. We wanted to implement this feature because once the workflow table is saved into the data model, the user can immediately view the results using our data visualization widget, rather than having to open MySQL or Neo4j or MongoDB to see the results separately. This makes it convenient for them to quickly receive the feedback and results of their workflow(s).

Also, for all of us, this was our first time using Python Dash to host an application, and it took some time getting used to how Dash works and how best to use its features or write code with it. We also needed to do research on how to connect our data models into Python in order to connect the frontend with backend logics.

Another conundrum we dealt with when trying to add human-in-the-loop features was *where*, exactly, a human can enter the loop. To design and implement this, we decided to look at current workflow systems and recent research papers documenting human-in-the-loop applications. Even when researching, it was difficult to pinpoint exactly where a human should be able to give input because many applications offer entirely different human-in-the-loop features, so there was no general consensus to be found. A more reasonable approach would have required user feedback, where we present the system to users to determine what feature they would have found most useful for human-in-the-loop interaction, but due to the time and resource constraints for this academic project, that was not an option. Ultimately, we decided to allow a person to specify not only the data model with which to define their workflow on, but also what closely-fitted data to keep as they saw fit when the workflow initiates.

We also had to consider how exactly we should represent workflows for the data set, as well as how to initiate them and create dependencies between them. The latter requires our application to keep track of existing dependencies between workflows, and automatically start a workflow whose dependency specifies that it should execute after another one is finished. During construction of the system, we must take various factors into consideration and carefully design the interaction of each function and component. Those elements include interruption from the UI, chaining workflows, timely updating sequences, and halting workflows for inspection.

## 4 Novel Design

For the MP, we treated Apache Airflow as an example of a state-of-the-art workflow model. To the best of our understanding, most other workflow applications are similar to Apache Airflow.

The main features that we have in our application which are different are the human-in-the-loop interaction and the capability of executing workflows based on workflow completion. In Apache Airflow, the workflow is only initiated by execution time, but we wanted to allow workflows to rely on the ending stage of other workflows so that, for example, when workflow A finishes, workflow B will automatically begin. We think that this is a useful feature because it simulates how real world tasks are conducted and satisfies users' usual and instant needs to extract/clean/analyze data.

In terms of why human-in-the-loop interaction has not been included in previous systems, it may be that other systems and Airflow are focused primarily on the automation and scheduling of workflows, and they prioritize this rather than human-computer interaction. It is the case that many existing systems leave as much as possible to the computer because, simply put, they do not trust the user. However, as many researchers working on human-in-the-loop applications will argue, a person can be helpful if they are given the sufficient knowledge and understanding of the system to succeed; many such systems, sadly, do not ease the user into the system and so they may be left confused as to what they are supposed to do. In this case, we developed an inspection mechanism during workflows: the system picks out data entries that do not strictly qualify the configured query conditions but have the potential to be equally meaningful, and then gather them in an intermediate data table for inspection. After the user has picked desired data from them, the workflow continues and concatenate these selected data with strictly qualified data. Our standard to determine whether or not a data entry deserves to be inspected is based on Reddit score. For future work, a recommendation/dedicated machine learning model would provide much more delicate results.

On the other hand, complex systems like Airflow rarely include support for inter-dependency between workflows. Each task entitles a specific execution time, bringing major inconvenience for users with light amounts of tasks to schedule. It is the case because the execution time and performance of workflows are nearly impossible to predict. If dependency on prior tasks coexist with strict time schedules, the system will likely take a massively queue and delayed execution. Fortunately for our light-weight system there is no such consideration. We managed to specify sequence of execution while maintaining strict schedules(but if a task is activated due to dependency on a previous task, its schedule will be reset). On a more industrial scale, the problem deserves to be more thoroughly evaluated. For example, one idea would to evaluate the execution time of a workflow with practice runs or model prediction, and allow systems to make recommendation to users about its schedule, or its dependent task's schedule.